Bottom-Up Syntax Analysis
LR - metódy

Ján Šturc

Zima 2010

Shift-reduce parsing

- LR methods (Left-to-right, Righ most derivation)
  - SLR, Canonical LR, LALR
- Other special cases:
  - Operator-precedence parsing
  - Backward deterministic parsing via string matching

Shift-Reduce Conflicts

Ambiguous grammar:
S → **if** E **then** S
| **if** E **then** S **else** S
| …

Stack	Input	Action
\$…	…\$	…
\$…**if** E **then** S	**else**…\$	shift or reduce?

Resolve in favor of shift, so **else** matches closest **if**.

Reduce-Reduce Conflicts

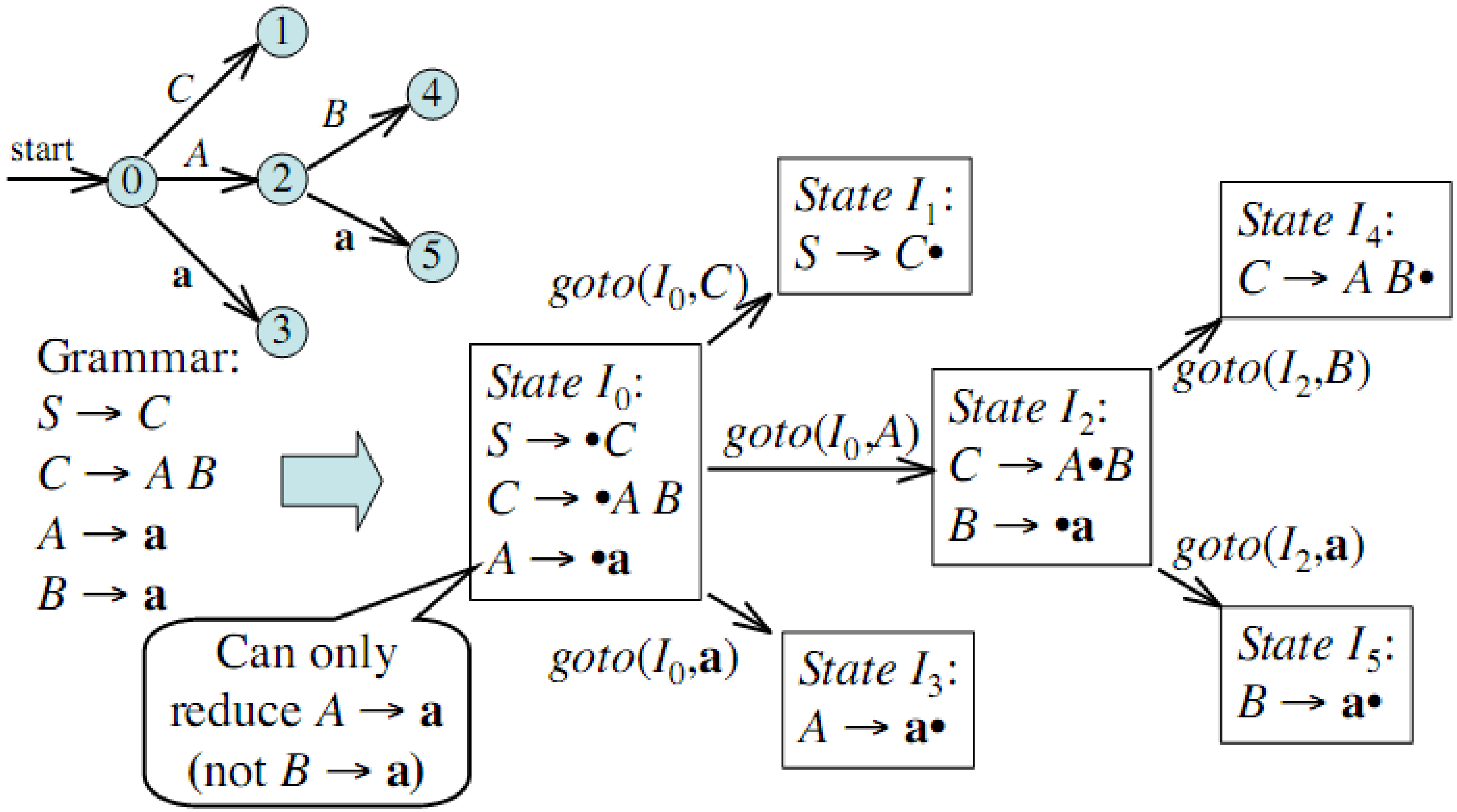
Grammar:
C → A B
A → **a**
B → **a**

Stack	Input	Action
$	**aa$**	shift
$a	**a$**	reduce A → **a** or B → **a** ?
$A	**a$**	shift
$A**a**	**$**	reduce A → **a** or B → **a** ?
$AB	**$**	reduce
$C	**$**	accept

Conflicts resolving on the base of the state (history) of the parsing.

In general, we can resolve conflicts on the state of the parsing, lookahead symbols , operator precedence, length of right-hand side of production.

LR(k) Parsers: Use a DFA for Shift/Reduce Decisions

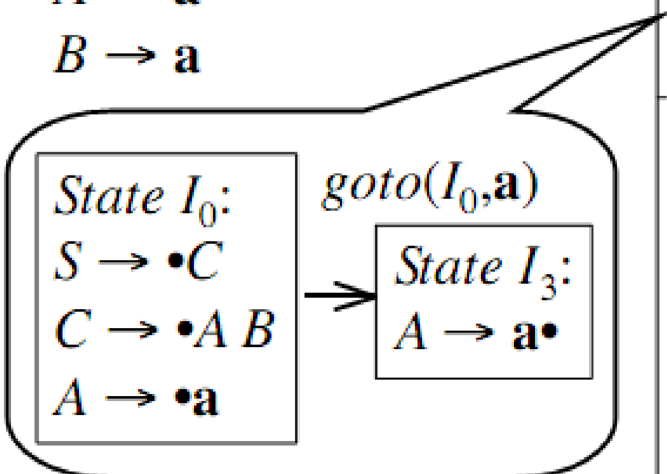


Položkový automat

DFA for Shift/Reduce Decisions

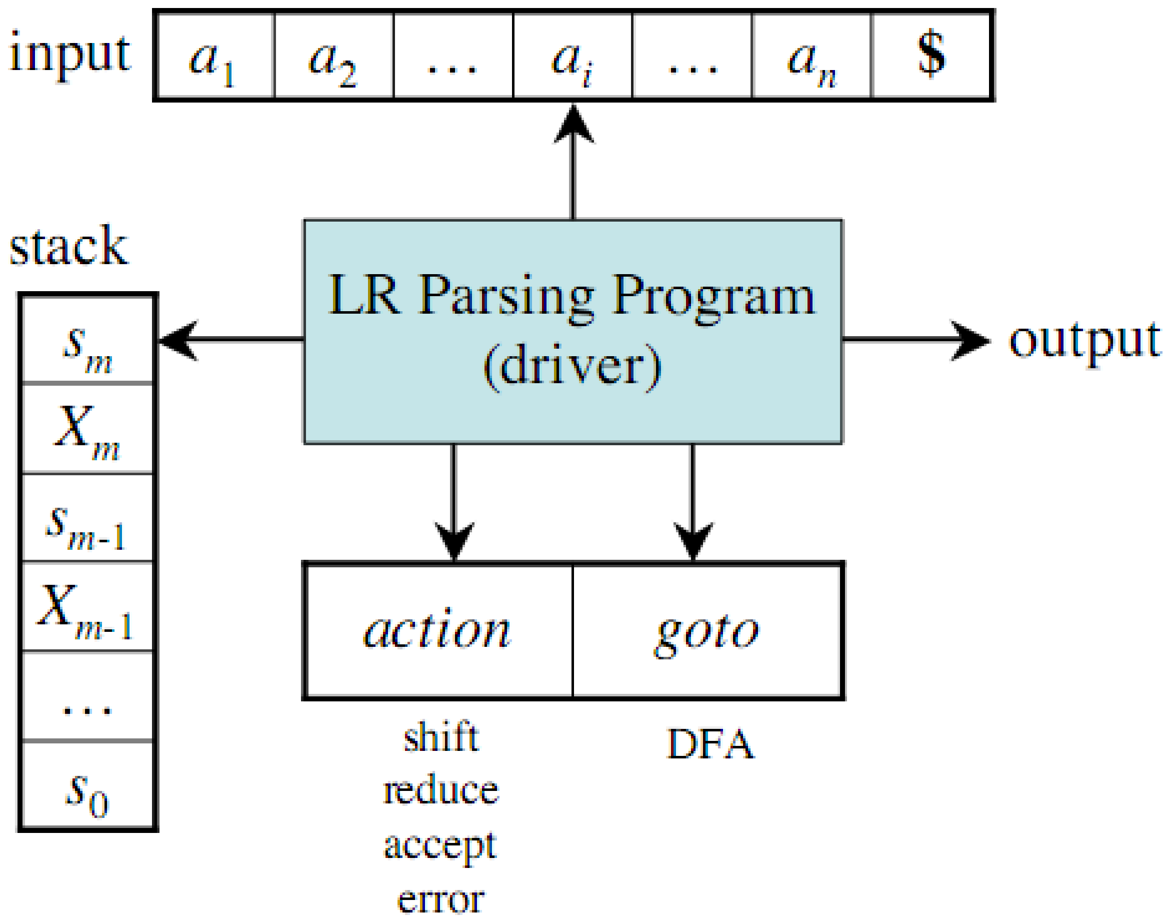
Grammar:
$S \to C$
$C \to A\ B$
$A \to \mathbf{a}$
$B \to \mathbf{a}$

The states of the DFA are used to determine if a handle is on top of the stack



Stack	Input	Action
$ 0	**aa$**	start in state 0
$ 0	**aa$**	shift (and goto state 3)
$ 0 $\underline{\mathbf{a}}$ 3	**a$**	reduce $A \to \mathbf{a}$ (goto 2)
$ 0 A 2	**a$**	shift (goto 5)
$ 0 A 2 $\underline{\mathbf{a}}$ 5	**$**	reduce $B \to \mathbf{a}$ (goto 4)
$ 0 $\underline{A}$ 2 $\underline{B}$ 4	**$**	reduce $C \to AB$ (goto 1)
$ 0 $\underline{C}$ 1	**$**	reduce $S \to C$
$ 0 S 1	**$**	accept

Model of an LR Parser



LR Parsing

Configuration (= LR parser state):

$$\underbrace{s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_m\ s_m,}_{\text{stack}}\quad \underbrace{a_i\ \ a_{i+1}\ \ldots\ a_n\ \$}_{\text{input}}$$

If action[s, a] = shift s, then push a , push s, and advance input:

$$s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_m\ s_m\ a_i\ \ s, \quad a_{i+1}\ \ldots\ a_n\ \$$$

If action[s_m, a_i] = reduce A → β and goto[s_{m-r}, A] = s with r=|β| then pop 2r symbols, push A, and push s:

$$s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_{m-r}\ s_{m-r}\ \ A\ s, \quad a_i\ a_{i+1}\ \ldots\ a_n\ \$$$

If action[s_m, a_i] = accept, then stop

If action[s_m, a_i] = error, then report the error and attempt to recovery

Example LR Parse Table

Grammar:

1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

	action						*goto*		
state	**id**	**+**	*****	**(**	**)**	**$**	*E*	*T*	*F*
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

shift 5

reduce by the production # 1

Example LR Parsing

Grammar:

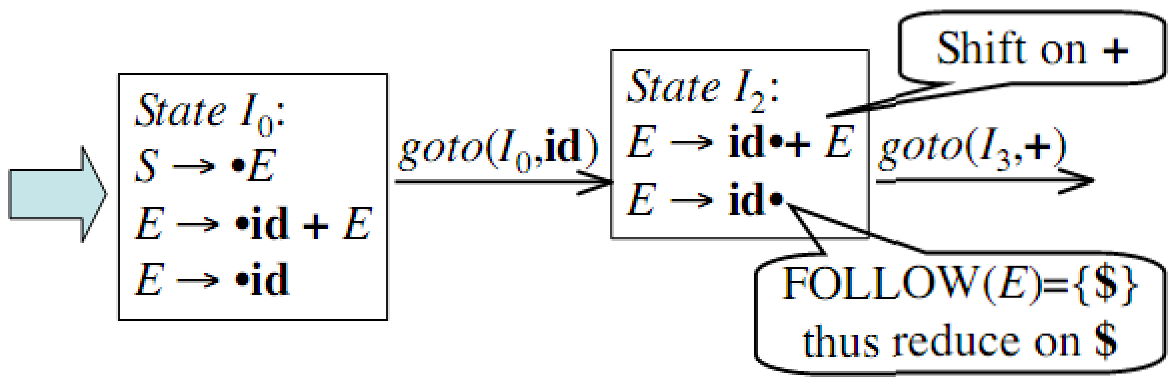
1. E → E + T
2. E → T
3. T → T * F
4. T → F
5. F → (E)
6. F → id

Stack	Input	Action
\$ 0	**id*id+id\$**	shift 5
\$ 0 **id** 5	***id+id\$**	reduce 6 goto 3
\$ 0 *F* 3	***id+id\$**	reduce 4 goto 2
\$ 0 *T* 2	***id+id\$**	shift 7
\$ 0 *T* 2 * 7	**id+id\$**	shift 5
\$ 0 *T* 2 * 7 **id** 5	**+id\$**	reduce 6 goto 10
\$ 0 *T* 2 * 7 *F* 10	**+id\$**	reduce 3 goto 2
\$ 0 *T* 2	**+id\$**	reduce 2 goto 1
\$ 0 *E* 1	**+id\$**	shift 6
\$ 0 *E* 1 **+** 6	**id\$**	shift 5
\$ 0 *E* 1 **+** 6 **id** 5	**\$**	reduce 6 goto 3
\$ 0 *E* 1 **+** 6 *F* 3	**\$**	reduce 4 goto 9
\$ 0 *E* 1 **+** 6 *T* 9	**\$**	reduce 1 goto 1
\$ 0 *E* 1	**\$**	accept

SLR parsing

- SLR (Simple LR): a simple extension of LR(0) shift-reduce parsing
- SLR eliminates some conflicts by populating the parsing table with reductions A→α on symbols in FOLLOW(A)

S → E
E → id + E
E → id



SLR Parsing Table

- Reductions do not fill entire rows
- Otherwise the same as LR(0)

1. $S \rightarrow E$
2. $E \rightarrow \textbf{id} + E$
3. $E \rightarrow \textbf{id}$

	id	**+**	**$**	*E*
0	s2			1
1			acc	
2		s3	r3	
3	s2			4
4			r2	

Shift on **+**

FOLLOW(E)={**$**}
thus reduce on **$**

SLR Parsing

- An LR(0) state is a set of LR(0) items. An LR(0) item is a production with a • (dot) in the right-hand side
- Build the LR(0) DFA by
  - Closure operation to construct LR(0) items
  - Goto operation to determine transitions
- Construct the SLR parsing table from the DFA
- LR parser program uses the SLR parsing table to determine shift/reduce operations

Constructing SLR Parsing Tables

1. Augment the grammar with $S' \rightarrow S\$$
2. Construct the set $C=\{I_0,I_1,\ldots,I_n\}$ of LR(0) items
3. If $[A\rightarrow\alpha\bullet a\beta] \in I_i$ and $goto(I_i, a)=I_j$ then set $action[i,a]=shift\ j$
4. If $[A\rightarrow\alpha\bullet] \in I_i$ then set $action[i, a]=reduce\ A\rightarrow\alpha$ for all $a \in FOLLOW(A)$ (apply only if $A\neq S'$)
5. If $[S'\rightarrow S\bullet]$ is in I_i then set $action[i, \$]=accept$
6. If $goto(I_i, A)=I_j$ then set $goto[i, A]=j$
7. Repeat 3-6 until no more entries added
8. The initial state i is the I_i holding item $[S'\rightarrow\bullet S\$]$

LR(0) Items of a Grammar

- An LR(0) item of a grammar G is a production of G with a • at some position of the right-hand side
- Thus, a production $A \rightarrow X\ Y\ Z$ has four items:

  $[A \rightarrow \bullet\ X\ Y\ Z]$

  $[A \rightarrow X \bullet Y\ Z]$

  $[A \rightarrow X\ Y \bullet Z]$

  $[A \rightarrow X\ Y\ Z \bullet]$

- Note that production $A \rightarrow \varepsilon$ has one item $[A \rightarrow \bullet]$
  - here bullet is simultaneously at the beginning and at the end

Constructing the set of LR(0) Items of a Grammar

1. The grammar is augmented with a new start symbol S' and production S'→S**$**.
2. Initially, set C = closure({[S'→•S **$**]}) (this is the start state of the DFA).
3. For each set of items I ∈ C and each grammar symbol X ∈ (N∪T) such that goto(I, X) ∉ C and goto(I, X) ≠ Ø, add the set of items goto(I, X) to C.
4. Repeat 3 until no more sets can be added to C.

The Closure Operation for LR(0) Items

1. Start with closure(I) = I
2. If $[A \rightarrow \alpha \bullet B\beta] \in$ closure(I) then for each production $B \rightarrow \gamma$ in the grammar, add the item $[B \rightarrow \bullet\gamma]$ to I if it is not already in I
3. Repeat 2 until no new items can be added

The Closure Operation
(Example)

Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E)$

$F \rightarrow id$

$closure(\{[E' \rightarrow \bullet E]\}) =$

$\{ [E' \rightarrow \bullet E] \}$

$\{ [E' \rightarrow \bullet E]$
$[E \rightarrow \bullet E + T]$
$[E \rightarrow \bullet T] \}$

Add $[E \rightarrow \bullet \gamma]$

$\{ [E' \rightarrow \bullet E]$
$[E \rightarrow \bullet E + T]$
$[E \rightarrow \bullet T]$
$[T \rightarrow \bullet T * F]$
$[T \rightarrow \bullet F] \}$

Add $[T \rightarrow \bullet \gamma]$

$\{ [E' \rightarrow \bullet E]$
$[E \rightarrow \bullet E + T]$
$[E \rightarrow \bullet T]$
$[T \rightarrow \bullet T * F]$
$[T \rightarrow \bullet F]$
$[F \rightarrow \bullet (E)]$
$[F \rightarrow \bullet \mathbf{id}] \}$

Add $[F \rightarrow \bullet \gamma]$

The Goto Operation for LR(0) Items

1. For each item $[A \rightarrow \alpha \bullet X\beta] \in I$, add the set of items closure($\{[A \rightarrow \alpha X \bullet \beta]\}$) to goto(I, X) if not already there
2. Repeat step 1 until no more items can be added to goto(I, X)
3. Intuitively, goto(I, X) is the set of items that are valid for the viable prefix γX when I is the set of items that are valid for γ

The Goto Operation (Example 1)

Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E)$

$F \rightarrow \mathbf{id}$

Suppose I = { [$E' \rightarrow \bullet\ E$]
[$E \rightarrow \bullet\ E\ \mathbf{+}\ T$]
[$E \rightarrow \bullet\ T$]
[$T \rightarrow \bullet\ T\ \mathbf{*}\ F$]
[$T \rightarrow \bullet\ F$]
[$F \rightarrow \bullet\ \mathbf{(}\ E\ \mathbf{)}$]
[$F \rightarrow \bullet\ \mathbf{id}$] }

Then goto(I, E) =
closure({[$E' \rightarrow E \bullet$, $E \rightarrow E \bullet + T$]})
= { [$E' \rightarrow E \bullet$],
[$E \rightarrow E \bullet + T$] }

The Goto Operation (Example 2)

Grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E)$

$F \rightarrow id$

Suppose

$I = \{ [E' \rightarrow E \bullet], [E \rightarrow E \bullet + T] \}$

Then $goto(I,+) = closure(\{[E \rightarrow E + \bullet\ T]\}) = \{ [E \rightarrow E + \bullet\ T],$

$[T \rightarrow \bullet\ T * F],$

$[T \rightarrow \bullet\ F],$

$[F \rightarrow \bullet\ (E)],$

$[F \rightarrow \bullet\ id] \}$

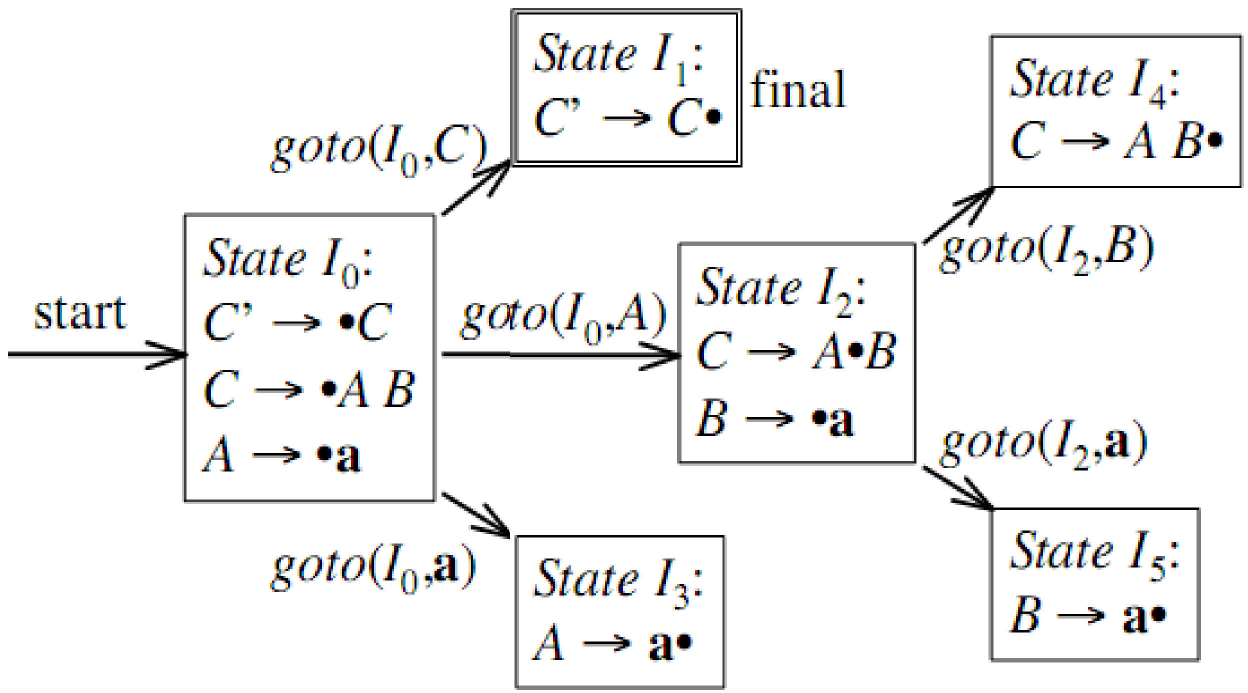
Example SLR Grammar and LR(0) Items

Augmented grammar:

1. C’ → C
2. C → A B
3. A → **a**
4. B → **a**

I_0 = closure({[C’ → •C]})

I_1 = goto(I0,C) = closure({[C’ → C•]})



Example SLR Parsing Table

State I_0:
$C' \to \bullet C$
$C \to \bullet A\ B$
$A \to \bullet \mathbf{a}$

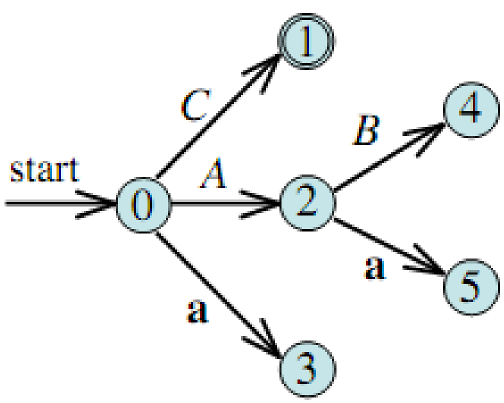
State I_1:
$C' \to C \bullet$

State I_2:
$C \to A \bullet B$
$B \to \bullet \mathbf{a}$

State I_3:
$A \to \mathbf{a} \bullet$

State I_4:
$C \to A\ B \bullet$

State I_5:
$B \to \mathbf{a} \bullet$



	a	**$**	*C*	*A*	*B*
0	s3		1	2	
1		acc			
2	s5				4
3	r3				
4		r2			
5		r4			

Grammar:
1. $C' \to C$
2. $C \to A\ B$
3. $A \to \mathbf{a}$
4. $B \to \mathbf{a}$

SLR and Ambiguity

- Every SLR grammar is unambiguous, but not every unambiguous grammar is SLR
- Consider for example the unambiguous grammar

1. $S \rightarrow L = R$
2. $S \rightarrow R$
3. $L \rightarrow * R$
4. $L \rightarrow \textbf{id}$
5. $R \rightarrow L$

I_0:
$S' \rightarrow \bullet S$
$S \rightarrow \bullet L{=}R$
$S \rightarrow \bullet R$
$L \rightarrow \bullet {*}R$
$L \rightarrow \bullet \textbf{id}$
$R \rightarrow \bullet L$

I_1:
$S' \rightarrow S \bullet$

I_2:
$S \rightarrow L \bullet {=} R$
$R \rightarrow L \bullet$

I_3:
$S \rightarrow R \bullet$

I_4:
$L \rightarrow {*} \bullet R$
$R \rightarrow \bullet L$
$L \rightarrow \bullet {*}R$
$L \rightarrow \bullet \textbf{id}$

I_5:
$L \rightarrow \textbf{id} \bullet$

I_6:
$S \rightarrow L{=}\bullet R$
$R \rightarrow \bullet L$
$L \rightarrow \bullet {*}R$
$L \rightarrow \bullet \textbf{id}$

I_7:
$L \rightarrow {*}R \bullet$

I_8:
$R \rightarrow L \bullet$

I_9:
$S \rightarrow L{=}R \bullet$

$action[2,=]=s6$
$action[2,=]=r5$

no

Has no SLR parsing table

LR(1) Grammars

- SLR too simple (málo využivajú lookahead, vlastne len pri redukcii FOLLOW)
- LR(1) parsing uses lookahead to avoid unnecessary conflicts in parsing table
- LR(1) item = LR(0) item + lookahead

LR(0) item:

$[A \rightarrow \alpha \bullet \beta]$

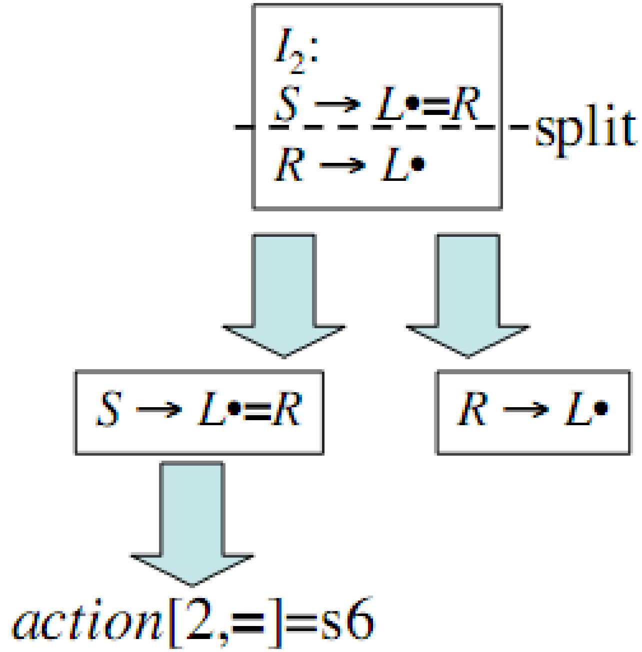
LR(1) item:

$[A \rightarrow \alpha \bullet \beta, a]$

- Obvykle sa LR(1) itemy implementujú ako trojica čísiel: číslo pravidla, pozícia bodky a vnútorný kód „lookahead tokenu“.

SLR Versus LR(1)

- Split the SLR states by adding LR(1) lookahead
- Unambiguous grammar
  1. S → L = R
  2. S → R
  3. L → * R
  4. L → **id**
  5. R → L



Should not reduce, because no right-sentential form begins with R=

LR(1) Items

- If an LR(1) item

  $[A \rightarrow \alpha \bullet \beta, a]$

  contains a lookahead terminal a, means α is already on the top of the stack and to see βa is expected.
- For items of the form

  $[A \rightarrow \alpha \bullet, a]$

  the lookahead a is used to reduce $A \rightarrow \alpha$ only if the next input is a
- For items of the form

  $[A \rightarrow \alpha \bullet \beta, a]$

  with $\beta \neq \varepsilon$ the lookahead has no effect during parsing

The Closure Operation for LR(1) Items

1. Start with closure(I) = I
2. If $[A \to \alpha \bullet B\beta, a] \in$ closure(I) then for each production $B \to \gamma$ in the grammar and each terminal $b \in \mathrm{FIRST}(\beta a)$, add the item $[B \to \bullet\gamma, b]$ to I if not already in I
3. Repeat 2 until no new items can be added

The Goto Operation for LR(1) Items

1. For each item $[A \rightarrow \alpha \bullet X\beta, a] \in I$, add the set of items closure($\{[A \rightarrow \alpha X \bullet \beta, a]\}$) to goto(I,X) if not already there
2. Repeat step 1 until no more items can be added to goto(I, X)

Constructing the set of LR(1) Items of a Grammar

1. Augment the grammar with a new start symbol S' and production $S' \rightarrow S\$$
2. Initially, set C = closure({[$S' \rightarrow \bullet S$, \$]}) this is the start state of the DFA.
3. For each set of items $I \in C$ and each grammar symbol $X \in (N \cup T)$ such that goto(I, X) $\notin$ C and goto(I, X) $\neq \emptyset$, add the set of items goto(I,X) to C
4. Repeat 3 until no more sets can be added to C

Example Grammar and LR(1) Items

- Unambiguous LR(1) grammar:
  2. S → L **=** R
  3. S → R
  4. L → ***** R
  5. L → **id**
  6. R → L
- Augment with 1. S' → S**$**
- LR(1) items (next slide)

I_0: [S' → •S, **$**] goto($I_0$,S)=$I_1$
[S → •L=R, **$**] goto($I_0$,L)=$I_2$
[S → •R, **$**] goto($I_0$,R)=$I_3$
[L → •*****R, **=**] goto(I_0,*****)=I_4
[L → •**id**, **=**] goto(I_0,**id**)=I_5
[R → •L, **$**] goto($I_0$,L)=$I_2$

I_1: [S' → S•, **$**]

I_2: [S → L•=R, **$**] goto($I_2$,**=**)=$I_6$
[R → L•, **$**]

I_3: [S → R•, $]

I_4: [L → *****•R, **=**] goto(I_4,R)=I_7
[R → •L, **=**] goto(I_4,L)=I_8
[L → •*R, **=**] goto(I_4,*****)=I_4
[L → •id, **=**] goto(I_4,id)=I_5

I_5: [L → id•, **=**]

I_6: [S → L=•R, **$**] goto($I_6$,R)=$I_4$
[R → •L, **$**] goto($I_6$,L)=$I_{10}$
[L → •*****R, **$**] goto($I_6$,*****)=$I_{11}$
[L → •id, **$**] goto($I_6$,**id**)=$I_{12}$

I_7: [L → *****R•, **=**|**$**]

I_8: [R → L•, **=**]

I_9: [S → L=R•, **$**]

I_{10}: [R → L•, **$**]

I_{11}: [L → *****•R, **$**] goto($I_{11}$,R)=$I_{13}$
[R → •L, **$**] goto($I_{11}$,L)=$I_{10}$
[L → •*****R, **$**] goto($I_{11}$,*****)=$I_{11}$
[L → •**id**, **$**] goto($I_{11}$,**id**)=$I_{12}$

I_{12}: [L → **id**•, **$**]

I_{13}: [L → *****R•, **$**]

Canonical LR(1) Parsing Tables

1. Augment the grammar with $S' \rightarrow S\$$
2. Construct the set $C=\{I_0,I_1,\ldots,I_n\}$ of LR(1) items
3. If $[A \rightarrow \alpha \bullet a\beta, b] \in I_i$ and $goto(I_i, a)=I_j$ then set $action[i, a]=shift\ j$ (b is here irrelevant)
4. If $[A \rightarrow \alpha \bullet, a] \in I_i$ then set (Surely $a \in Follow(A)$) $action[i, a]=reduce\ A \rightarrow \alpha$ (apply only if $A \neq \$$)
5. If $[S' \rightarrow S \bullet, \$]$ is in I_i then set $action[i, \$]=accept$
6. If $goto(I_i, A)=I_j$ then set $goto[i, A]=j$
7. Repeat 3-6 until no more entries added
8. The initial state i is the I_i holding item $[S' \rightarrow \bullet S,\$]$

Example LR(1) Parsing Table

Grammar:

1. S' → S **$**
2. S → L **=** R
3. S → R
4. L → ***** R
5. L → **id**
6. R → L

	id	*****	**=**	**$**	*S*	*L*	*R*
0	s5	s4			1	2	3
1				acc			
2			s6	r6			
3				r3			
4	s5	s4				8	7
5			r5	r5			
6	s12	s11				10	4
7			r4	r4			
8			r6	r6			
9				r2			
10				r6			
11	s12	s11				10	13
12				r5			
13				r4			

LALR(1) Grammars

- LR(1) parsing tables have too many states
- LALR(1) parsing (Look-Ahead LR) merges LR(1) states to reduce table size
- Less powerful than LR(1)
  - Will not introduce shift-reduce conflicts, because shifts do not use lookaheads
  - May introduce reduce-reduce conflicts, but seldom do so for grammars of programming languages

Constructing LALR(1) Parsing Tables

- Construct sets of LR(1) items
- Merge LR(1) sets with sets of items that share the same first part Kernel item ([S → •S$] or • is inside right-hand side)

I_4:	[L → *•R,	=]
	[R → •L,	=]
	[L → •*R,	=]
	[L → •id,	=]
I_{11}:	[L → *•R,	$]
	[R → •L,	$]
	[L → •*R,	$]
	[L → •**id**,	$]

Shorthand for two items

[L → *•R,	=|$]
[R → •L,	=|$]
[L → •*R,	=|$]
[L → •id,	=|$]

Example LALR(1) Grammar

- Augmented unambiguous LR(1) grammar:
  1. S’ → S **$**
  2. S → L **=** R
  3. S → R
  4. L → ***** R
  5. L → **id**
  6. R → L
- LALR(1) items (next slide)

I_0:	$[S' \rightarrow \bullet S,$	$\$]$ goto$(I_0,S)=I_1$
	$[S \rightarrow \bullet L{=}R,$	$\$]$ goto$(I_0,L)=I_2$
	$[S \rightarrow \bullet R,$	$\$]$ goto$(I_0,R)=I_3$
	$[L \rightarrow \bullet{*}R,$	$=]$ goto$(I_0,*)=I_4$
	$[L \rightarrow \bullet\mathbf{id},$	$=]$ goto$(I_0,\mathbf{id})=I_5$
	$[R \rightarrow \bullet L,$	$\$]$ goto$(I_0,L)=I_2$
I_1:	$[S' \rightarrow S\bullet,$	$\$]$
I_2:	$[S \rightarrow L\bullet{=}R,$	$\$]$ goto$(I_0,=)=I_6$
	$[R \rightarrow L\bullet,$	$\$]$
I_3:	$[S \rightarrow R\bullet,$	$\$]$
I_4:	$[L \rightarrow {*}\bullet R,$	$=/\$]$ goto$(I_4,R)=I_7$
	$[R \rightarrow \bullet L,$	$=/\$]$ goto$(I_4,L)=I_9$
	$[L \rightarrow \bullet{*}R,$	$=/\$]$ goto$(I_4,*)=I_4$
	$[L \rightarrow \bullet\mathbf{id},$	$=/\$]$ goto$(I_4,\mathbf{id})=I_5$
I_5:	$[L \rightarrow \mathbf{id}\bullet,$	$=/\$]$

I_6:	$[S \rightarrow L{=}\bullet R,$	$\$]$ goto$(I_6,R)=I_8$
	$[R \rightarrow \bullet L,$	$\$]$ goto$(I_6,L)=I_9$
	$[L \rightarrow \bullet{*}R,$	$\$]$ goto$(I_6,*)=I_4$
	$[L \rightarrow \bullet\mathbf{id},$	$\$]$ goto$(I_6,\mathbf{id})=I_5$
I_7:	$[L \rightarrow {*}R\bullet,$	$=/\$]$
I_8:	$[S \rightarrow L{=}R\bullet,$	$\$]$
I_9:	$[R \rightarrow L\bullet,$	$=/\$]$

Shorthand for two items

$[R \rightarrow L\bullet, \quad =]$
$[R \rightarrow L\bullet, \quad \$]$

Example LALR(1) Parsing Table

Grammar:

1. S' → S **$**
2. S → L **=** R
3. S → R
4. L → ***** R
5. L → **id**
6. R → L

	id	*****	**=**	**$**	*S*	*L*	*R*
0	s5	s4			1	2	3
1				acc			
2			s6	r6			
3				r3			
4	s5	s4				9	7
5			r5	r5			
6	s5	s4				9	8
7			r4	r4			
8				r2			
9			r6	r6			

Improving of the LALR(1) state construction

- Disadvantage of the previous constuction is that during construction all the LR(1) states are created and then merged to LR(0) states.
- If we neglect the ε-production, the reductions take place for kernel items only.
- Reduction $A \rightarrow \varepsilon$ is called on input a iff there is a kernel item $[B \rightarrow \beta \bullet C\gamma, b]$ such that $C \Rightarrow_{rm} A\delta$ and a is in First($\delta\gamma b$). So the ε-reductions for the kernel items can be precomputed.
- In the computation of the closure of an item $[B \rightarrow \gamma \bullet \delta, a]$ there are only two possibilities for lookahead symbols for the items in the closure; they arise according to step 2 at slide 32, we say the are generated spontaneously or it is a and we say that a propagate.

Computation of the lookaheads

for each item $B \rightarrow \gamma \cdot \delta$ in K **do**

  begin

    $J' := \text{CLOSURE}(\{[B \rightarrow \gamma \cdot \delta,\ \#]\})$

    if $[A \rightarrow \alpha \cdot X \beta,\ a]$ is in J', where a is not #, **then**

      lookahead a is generated spontaneously for item

      $A \rightarrow \alpha X \cdot \beta$ in $\text{GOTO}(I,\ X)$;

    if $[A \rightarrow \alpha \cdot X \beta,\ \#]$ is in J', **then**

      lookaheads propagate from $B \rightarrow \gamma \cdot \delta$ in I to

      $A \rightarrow \alpha X \cdot \beta$ in $\text{GOTO}(I,\ X)$

  end

LALR(1) sets of items construction

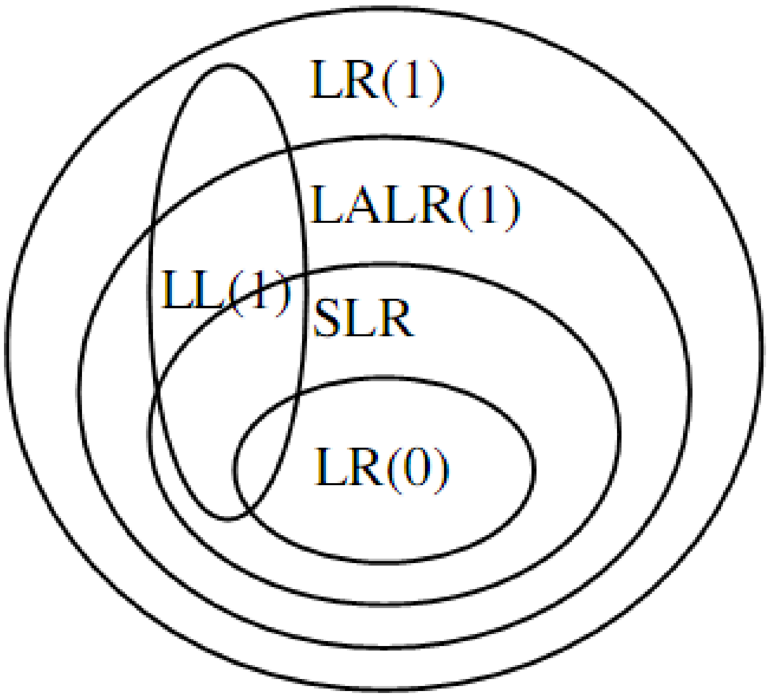
```
procedure INSERT(I, A → α·β, a);
if not ON[I, A → α·β, a] then
    begin
        push (I, A → α·β, a) onto STACK;
        ON[I, A → α·β, a] := true;
        add a to the list of lookaheads for A → α·β in I
    end;

begin
(1)     for all I, A → α·β and a do ON[I, A → α·β, a] := false;
(2)     STACK := empty;
(3)     INSERT(I_0, S' → ·S, $);
(4)     for each I, A → α·β and a such that a is spontaneously
          generated as a lookahead for A → α·β in I do
(5)         INSERT(I, A → α·β, a);
(6)     while STACK not empty do
            begin
(7)             pop (J, B → γ·δ, a), the top triple
                    on STACK off of STACK;
(8)             for each grammar symbol X do
(9)                 for each A → α·β in the kernel of GOTO(J, X)
                        such that B → γ·δ in J propagates lookaheads to
                        A → α·β in GOTO(J, X) do
(10)                        INSERT(GOTO(J, X), A → α·β, a)
            end
end
```

Error treatment

- Empty fields of the action table are places for insertion the error reporting and recovery action
  - Synch (panic mode)
  - Phrase level recovery
- LR like analysis (LR, SLR, LALR) discovers errors as soon as possible, whenever the parsed string is not a viable prefix any word generated by the given grammar

LL, SLR, LR, LALR Grammars



- Most of the programming languages have LALR(1) or even SLR(1) grammar.
- All the deterministic CF-languages can be covered by the $\bigcup_i LR(i)$.
- The compiler generators, TWS's and compiler compilers including Bison and Yacc use mostly LALR(1) analysis